

# The Project Suite



## Streamlined QMC Workflows

Jaron T. Krogel  
11 December 2013

# Components/Stages of QMC Projects

## Selecting atomic structure

- generating/manipulating structure
- DFT structural relaxation

## Selecting cell size/k-points

- DFT total energy vs cell size/k-points
- DMC total energy vs cell size

## Selecting wavefunction

- DFT orbital generation (scf/nscf)
- wavefunction conversion
- VMC Jastrow optimization

## Selecting QMC parameters

- meshfactor convergence
- timestep study

## Selecting pseudopotential

- DFT crystal equation of state
- DFT crystal phonon spectrum
- DMC atomic ionization potential
- DMC dimer bonding

## Production DMC runs

- equation of state
- defect formation energy
- excitations
- surface adsorbtion

## For all of these

- write input files
- submit/monitor jobs
- analyze output data
- chain info between jobs

# Simplifying Project Management

## The Project Suite

- main idea: make each project component easier to execute
- high-level scripting environment to manage arbitrary workflows
- input files, simulations, and output data represented as high-level objects
- dependencies between simulations specified to define workflows
- job submission and data flow bet. chained jobs handled automatically
- processed output data available for interactive or scripted access

## Advantages

- researcher can focus on project design and interpreting results
- removes opportunities for error propagation
- implement and share new workflows involving various codes
- automatic record of project process, easy to reproduce by anyone else
- easier for those new to QMC to immediately begin productive work

# Working with the Project Suite

## User Interface

- basic use through input file like python script
- more advanced workflows with looping & logic constructs

## Typical stages

- specify settings: PP directory, job monitoring frequency, ...
- generate structure or read from file
- create simulation objects from minimal set of inputs
- specify data dependencies between simulations
  - e.g. structure, charge density, orbitals, jastrow, ...
- pass simulation objects to ProjectManager, have it perform all runs
- load data analysis objects and extract energies, make plots, etc.

```
#!/usr/bin/env python

from project import settings, ProjectManager, Job
from project import generate_physical_system
from project import generate_pwscf

settings(
    pseudo_dir      = './pseudopotentials',
    generate_only   = 1,
    status_only     = 0,
    machine        = 'node16'
)

diamond = generate_physical_system(
    lattice      = 'cubic',
    cell         = 'conventional',
    centering    = 'F',
    constants    = 3.57,
    units        = 'A',
    atoms        = 'C',
    basis        = [[0,0,0],[.25,.25,.25]],
    tiling       = (2,2,2),
    C            = 4
)

scf = generate_pwscf(
    identifier   = 'scf',
    path         = 'dft_diamond',
    job          = Job(cores=12),
    input_type   = 'scf',
    input_dft    = 'lda',
    ecut         = 200,
    conv_thr    = 1e-6,
    mixing_beta = .7,
    pseudos     = ['C.BFD.upf'],
    system      = diamond,
    kgrid       = (2,2,2),
    kshift      = (1,1,1)
)

pm = ProjectManager()
pm.add_simulations(scf)
pm.run_project()

pa = scf.load_analyzer_image()
print 'DFT total energy is: ', pa.E
```

← Project Suite imports

← Project Suite settings

← structure generation

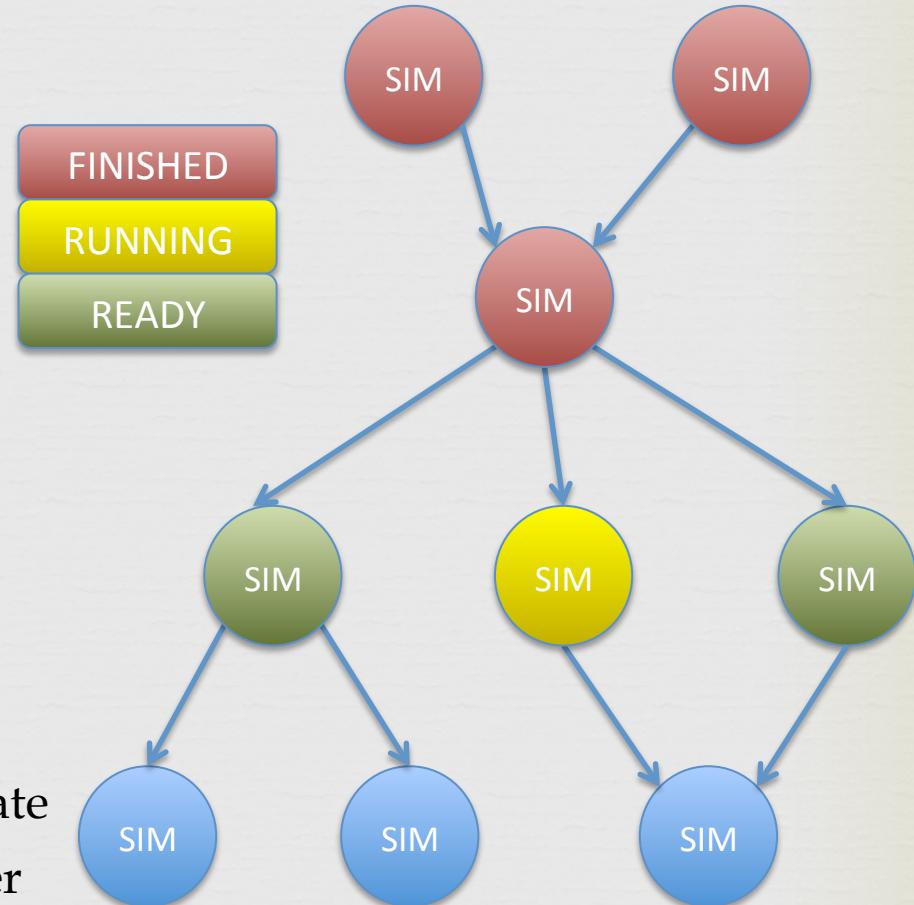
← PWSCF simulation

← job submission

← data analysis

# The Project Manager

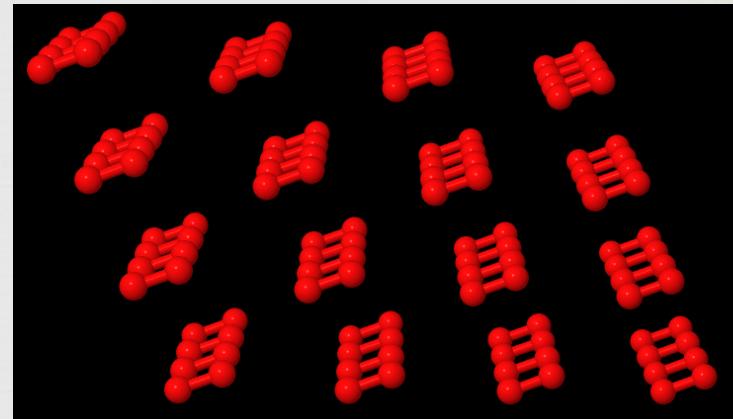
- takes a list of simulation objects
- workflow network from dependencies
- periodically walks down network
  - writes input files
  - submits and monitors jobs
  - analyzes output for new input
  - sleeps in between checks
- manages many workflows in parallel
- remembers simulation status
  - picks up where it left off
  - does not resubmit previous jobs
- copies light input/output data to separate directory structure for easy file transfer



# Structure Generation/Manipulation

- Structure generator for crystals (lattice + basis)

```
02_prim = generate_structure(  
    lattice      = 'monoclinic',  
    cell         = 'primitive',  
    centering    = 'C',  
    constants    = (5.403,3.429,5.086,132.53),  
    units        = 'A',  
    angular_units = 'degrees',  
    atoms        = ('O','O'),  
    basis         = [[0,0,1.15/2],[0,0,-1.15/2]],  
    basis_vectors = identity(3)  
)
```



- Create supercells w/ integer or matrix tiling

```
02_super = 02_prim.tile(4,4,4)
```

- Read/write .xyz files

```
02_prim.write_xyz('./02_prim.xyz')  
02_super.write_xyz('./02_super.xyz')
```

# Structure Generation/Manipulation

## Cell operations

- Tile, fold, rescale, or skew cell
- Calculate volume, Madelung constant, Wigner radius of cell

## k-points operations

- Add k-points individually or as Monkhorst-Pack mesh
- Reduce to unique k-points under inversion symmetry

## Atom/position operations

- Select, replace, remove, translate, or freeze atoms
- Calculate bonds, distance tables, neighbor tables, neighbor shells

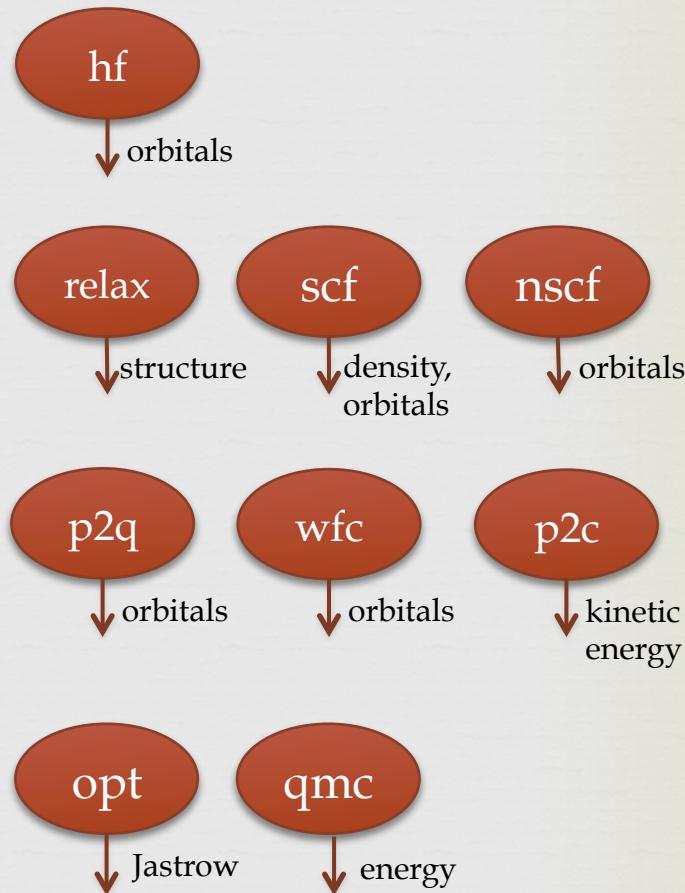
## Future operations

- Apply or discover symmetry operations
- Interface with structural databases

# Workflow Components (Simulations)

## Orbital/structure sources

- SQD - numerical Hartree-Fock code for atoms
- PWSCF - DFT orbitals & relaxed structures



## Orbital conversion & information

- pw2qmcpack, wfconvert, pw2casino

## Jastrow optimization/VMC/DMC

- QMCPACK - optimization & total energy

# PWSCF Interface

- Simulation generate function, small number of inputs

simulation object → scf = generate\_pwscf(  
  identifier = 'file\_prefix',  
  path = 'diamond/dft',  
  job = Job(cores=12),  
  input\_type = 'scf', ←  
  input\_dft = 'lda',  
  ecut = 200,  
  conv\_thr = 1e-6,  
  mixing\_beta = .7,  
  pseudos = ['C.BFD.upf'],  
  kgrid = (2,2,2),  
  kshift = (1,1,1),  
  system = diamond ←  
)

generic job info

mode selector  
(scf, nscf, or relax)

pwscf inputs

physical system object

# PWSCF Interface

- Separate inputs from sim. generation for greater flexibility

inputs  
storage  
object

```
→ scf_inputs = obj(  
    identifier = 'file_prefix',  
    path       = 'diamond/dft',  
    job        = Job(cores=12),  
    input_type = 'scf',  
    input_dft  = 'lda',  
    ecut       = 200,  
    conv_thr   = 1e-6,  
    mixing_beta = .7,  
    pseudos    = ['C.BFD.upf'],  
    kgrid      = (2,2,2),  
    kshift     = (1,1,1),  
    system     = diamond  
)
```

simulation → scf = generate\_pwscf(\*\*scf\_inputs)  
object

# Orbital Conversion Interface

- same information as input file, but more useful for workflows

```
p2q_inputs = obj(  
    identifier    = 'p2q',  
    path          = 'diamond/dft',  
    job           = Job(cores=1),  
    write_psir    = False  
)  
  
p2q = generate_pw2qmcpack(**p2q_inputs)
```

# QMCPACK Interface

## Jastrow Optimization

```
opt_inputs = obj(
    identifier = 'opt',
    path      = 'diamond/opt',
    job       = Job(cores=16),
    system    = system,
    input_type = 'opt_jastrow',
    pseudos   = ['C.BFD.xml'],
    jastrows  = 'generateJ12',
    opt_calcs = [
        linear(
            energy              = 0.0,
            unreweightedvariance = 1.0,
            reweightedvariance   = 0.0,
            samples              = 5e4,
            stepsbetweensamples = 10,
            warmupsteps          = 100,
            timestep             = 0.4,
            minmethod            = 'quartic'
        )
    ]
)

opt = generate_qmcpack(**opt_inputs)
```

## VMC and DMC

```
qmc_inputs = obj(
    identifier = 'dmc',
    path      = 'diamond/qmc',
    job       = Job(cores=16),
    system    = system,
    input_type = 'basic',
    pseudos   = ['C.BFD.xml'],
    calculations = [
        vmc(warmupsteps = 20,
            blocks     = 200,
            steps      = 10,
            substeps   = 3,
            timestep   = 0.3,
            samples    = 1000
        ),
        dmc(warmupsteps = 24,
            blocks     = 200,
            steps      = 10,
            timestep   = 0.01
        )
    ]
)

qmc = generate_qmcpack(**qmc_inputs)
```

# Workflow Examples

- Chained Structural Relaxation
- Optimization & DMC
- DFT Functional Scan
- DMC Equation of State

# Chained Structural Relaxation

- Relaxation w/ increasing Ecutf followed by SCF

```
ecuts = [30,40,50]

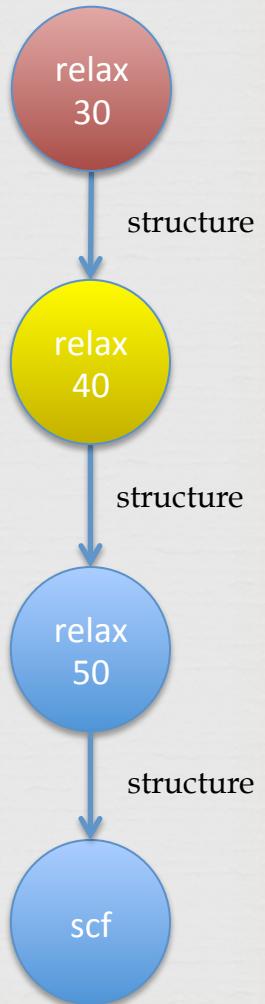
source = None
for ecut in ecuts:

    relax_inputs.ecut = ecut
    relax = generate_pwscf(**relax_inputs)

    if source!=None:
        relax.depends(source,'structure')
    #end if
    source = relax

#end for

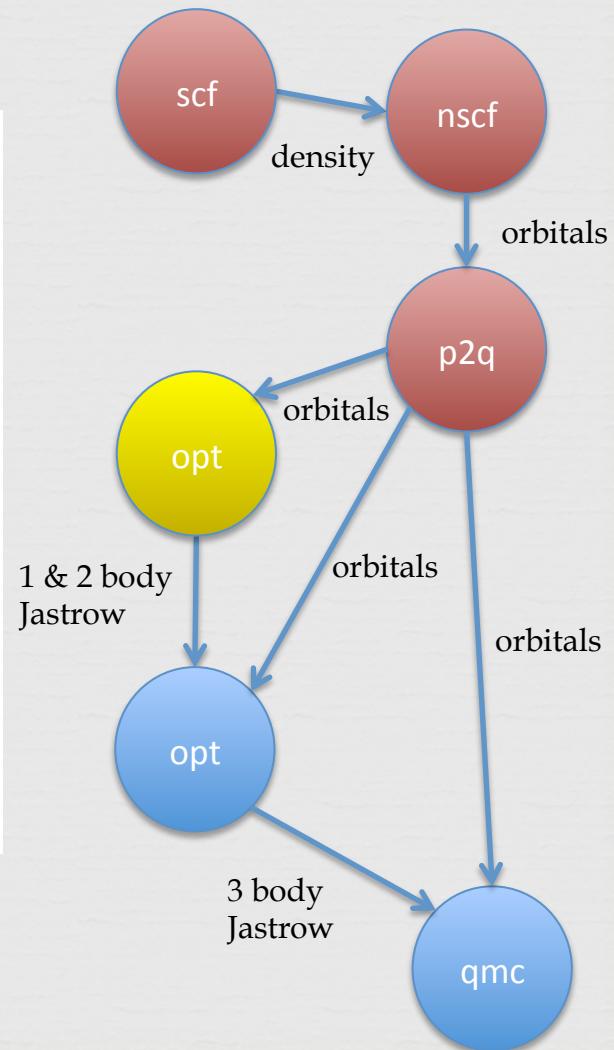
scf = generate_pwscf(**scf_inputs)
scf.depends(relax,'structure')
```



# Optimization & DMC

```
scf = generate_pwscf(**scf_inputs)
nscf = generate_pwscf(**nscf_inputs)
p2q = generate_pw2qmcpack(**p2q_inputs)
opt12= generate_qmcpack(**opt12_inputs)
opt3 = generate_qmcpack(**opt3_inputs)
qmc = generate_qmcpack(**qmc_inputs)

nscf.depends(scf, 'charge_density')
p2q.depends(nscf, 'orbitals')
opt12.depends(p2q, 'orbitals')
opt3.depends((p2q, 'orbitals'),
             (opt12, 'jastrow'))
qmc.depends((p2q, 'orbitals'),
            (opt3, 'jastrow'))
```



# DFT Functional Scan

```
functionals = ['lda', 'pbe', 'hse']

relax = generate_pwscf(**relax_inputs)

for functional in functionals:

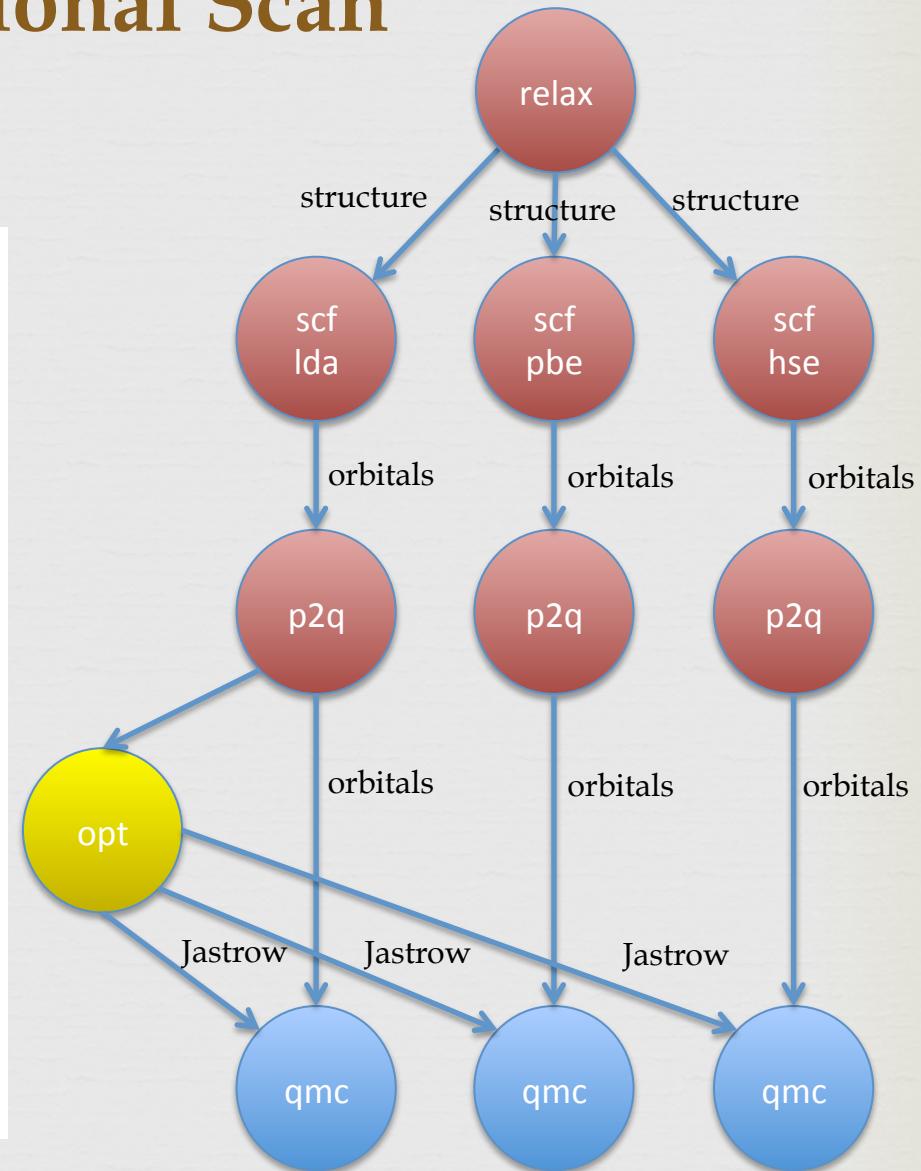
    scf_inputs.input_dft = functional
    scf = generate_pwscf(**scf_inputs)

    p2q = generate_pw2qmcpack(**p2q_inputs)

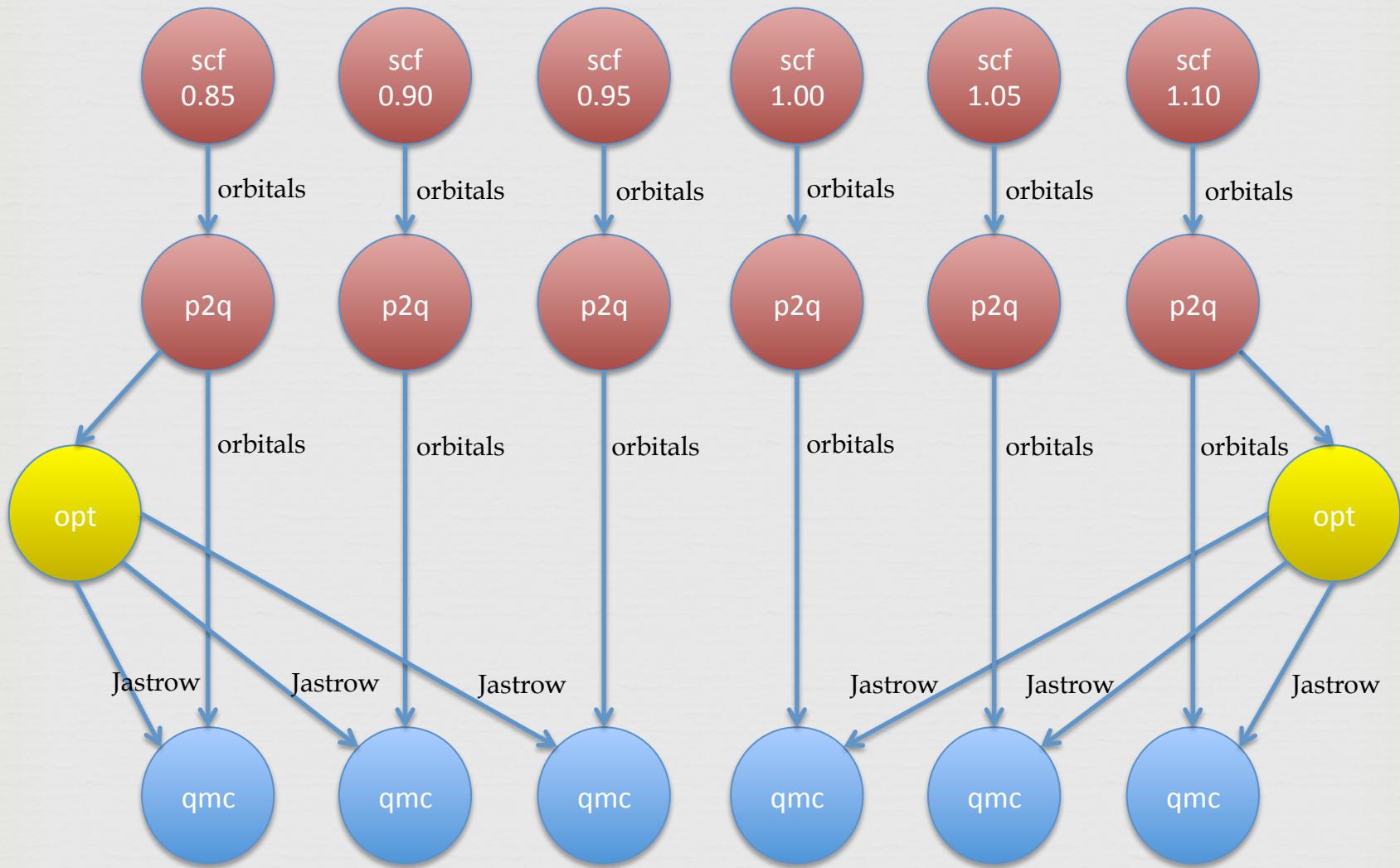
    if functional=='lda':
        opt = generate_qmcpack(**opt_inputs)
        opt.depends(p2q, 'orbitals')
    #end if

    qmc = generate_qmcpack(**qmc_inputs)

    scf.depends(relax, 'structure')
    p2q.depends(scf, 'orbitals')
    qmc.depends((p2q, 'orbitals'),
                (opt, 'jastrow'))
#end for
```



# DMC Equation of State



# DMC Equation of State

```
rescalings = [0.85,0.90,0.95,1.10,1.05,1.00]

for scale in rescalings:

    system = generate_physical_system(**system_inputs)
    system.structure.rescale(scale)

    scf_inputs.system = system
    qmc_inputs.system = system

    scf = generate_pwscf(**scf_inputs)

    p2q = generate_pw2qmcpack(**p2q_inputs)

    if scale==0.85 or scale==1.10:
        opt = generate_qmcpack(**opt_inputs)
        opt.depends(p2q,'orbitals')
    #end if

    qmc = generate_qmcpack(**qmc_inputs)

    p2q.depends(scf,'orbitals')
    qmc.depends((p2q,'orbitals'),
                (opt,'jastrow'))
#end for
```

# Workflow Management

```
pm = ProjectManager()

# Chained Structural Relaxation
pm.add_simulations(relax30,relax40,relax50,scf)

# Optimization & DMC
pm.add_simulations(scf,nscf,p2q,opt12,opt3,qmc)

# DFT Functional Scan
pm.add_simulations(relax,3*[scf,p2q,qmc],opt)

# DMC Equation of State
pm.add_simulations(6*[scf,p2q,qmc],2*[opt])

pm.run_project()
```

# Submission on Different Machines

## Workstation

```
settings(  
    machine = 'ws16'  
)  
  
sim=generate_xxxxxx(  
    job = Job(  
        cores  = 8,  
        threads = 4  
    )  
)
```

## Taub

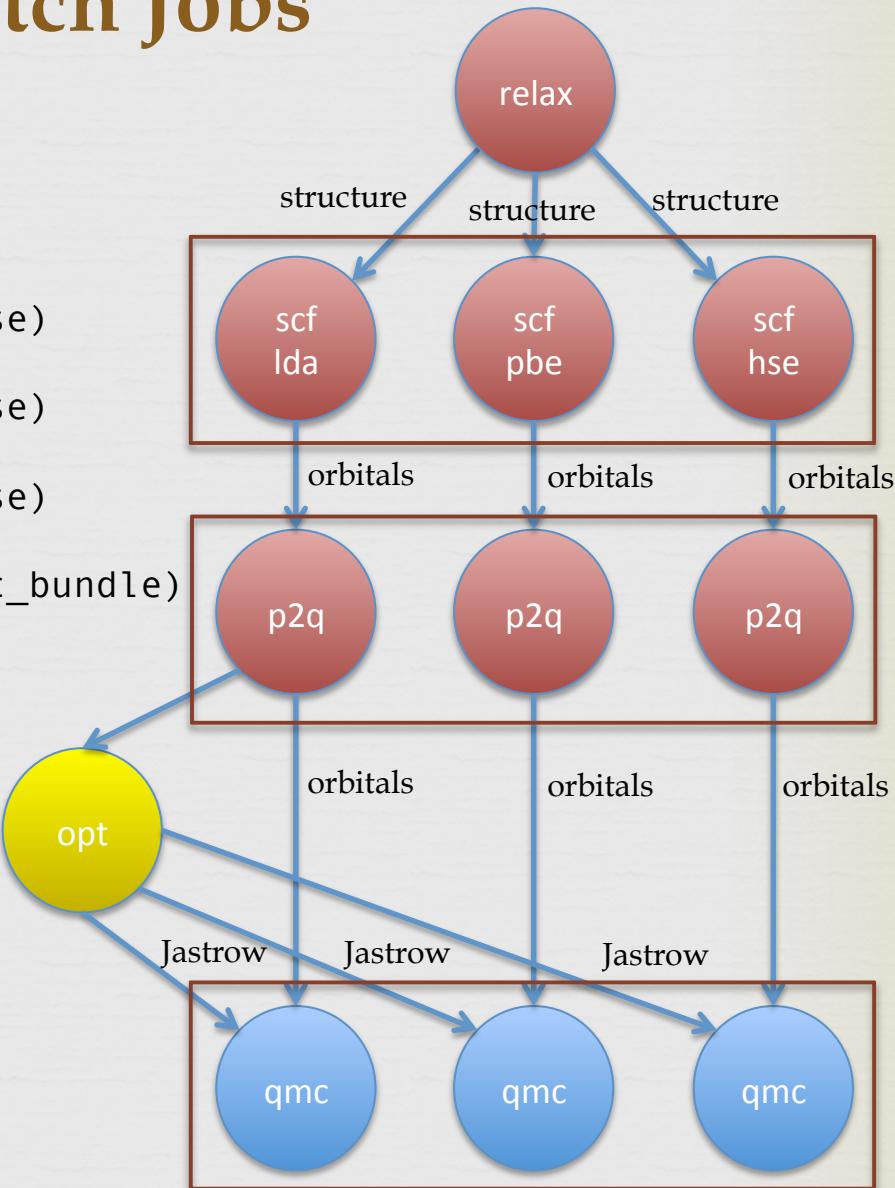
```
settings(  
    machine = 'taub'  
)  
  
sim=generate_xxxxxx(  
    job = Job(  
        nodes   = 12,  
        threads = 4,  
        hours   = 8,  
        minutes = 30  
    )  
)
```

## Titan

```
settings(  
    machine = 'titan',  
    account = 'XYZ123'  
)  
  
sim=generate_xxxxxx(  
    job = Job(  
        nodes   = 1000,  
        threads = 4,  
        hours   = 0,  
        minutes = 30,  
        queue   = 'debug'  
    )  
)
```

# Bundling Batch Jobs

```
scf_bundle = bundle(scf_lda, scf_pbe, scf_hse)
p2q_bundle = bundle(p2q_lda, p2q_pbe, p2q_hse)
qmc_bundle = bundle(qmc_lda, qmc_pbe, qmc_hse)
pm.add_simulations(scf_bundle,p2q_bundle,qmc_bundle)
```



# Bundling Batch Jobs

```
#!/bin/bash
#PBS -q some_queue
#PBS -N bundle
#PBS -l walltime=08:00:00
#PBS -l nodes=12:ppn=8
#PBS -W x="NACCESSPOLICY:SINGLEJOB"
#PBS -o bundle.out
#PBS -e bundle.err
#PBS -V

echo $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

date
export OMP_NUM_THREADS=4

cd /home/jtkrogel/runs/diamond/dft_lda
mpirun -np 32 pw.x -input scf.in >scf.out 2>scf.err&

cd /home/jtkrogel/runs/diamond/dft_pbe
mpirun -np 32 pw.x -input scf.in >scf.out 2>scf.err&

cd /home/jtkrogel/runs/diamond/dft_hse
mpirun -np 32 pw.x -input scf.in >scf.out 2>scf.err&

wait
```

# QMCPACK Data Analysis

## Capabilities

- statistical analysis of scalars.dat, dmc.dat, stat.h5
- plots of scalar traces, jastrow functions through optimization
- analysis of energy density
- twist averaging for all quantities

## Sample Usage

```
>>> qa=QmcpackAnalyzer('./qmc.in.xml',
                      analyze=True)

>>> qa.qmc
0                         VmcAnalyzer
1                         DmcAnalyzer
2                         DmcAnalyzer

>>> qa.qmc[2]
dmc                       DmcDatAnalyzer
info                      QAinformation
scalars                    ScalarsDatAnalyzer
scalars_hdf                ScalarsHDFAnalyzer
```

```
>>> qa.qmc[2].scalars_hdf
Coulomb                  obj
ElecElec                 obj
Kinetic                  obj
LocalEnergy               obj
LocalEnergy_sq            obj
LocalPotential            obj
data                     QAHDfdata

>>> print qa.qmc[2].scalars_hdf.LocalEnergy
error                   = 0.0201256357883
kappa                   = 12.5422841447
mean                    = -75.0484800012
sample_variance          = 0.00645881103012
variance                = 0.850521272106
```

# Obtaining the Project Suite

## Packaged with QMCPACK

```
export QMCPACK=https://subversion.assembla.com/svn/qmcdev/trunk  
svn checkout $QMCPACK/project_suite
```

## Setup is simple

- export PYTHONPATH=/your/path/to/project\_suite/library/
- requires python installation + numpy
- optional h5py for HDF5 files

## Documentation available

\$QMCPACK/project\_suite/documentation/project\_suite\_user\_guide.pdf

## Documentation covers

- walkthrough style tutorials on project suite use
- overview of QMC from a practitioner's perspective
- help for installing simulation codes and python libraries

# How to Extend It

- base classes for generic input files, simulations, output data, machines

## Adding a new batch machine

- derive from Supercomputer
- specify population of job sub. file (~20 lines)



## Adding a new simulation code

- input file class: read & write input file
- data analysis class: read & process output files
- simulation class: flow of data to & from self



# Summary

- Created suite of tools to make QMC research easier & more productive

## The Project Suite covers

- structure generation/manipulation
- minimal interfaces to simulation input files
- ability to chain simulations together into arbitrary workflows
- automatic job/workflow management on workstations & clusters
- simple interface to output data, including HDF5

## With sufficient interest

- developer contributions for using new QMCPACK features
- support for more codes (e.g. GAMESS) and features
- standard workflow sub-components
  - e.g. Ecut, k-point, meshfactor convergence
    - a tool to automate all-electron/PP IP's already exists
- command line tools
- interface to upcoming QMC database
- ability to manage workflows spanning machines from a workstation